APPLICATION NOTE

AP-162

December 1983

PMT Tutorial

BRIAN VALENTINE/JOHN JARVE
DSSO APPLICATIONS ENGINEERING

Order Number: 230879-001

INTRODUCTION

Intel's Program Management Tools (PMT's) provide the essential ingredients for managing software development projects. Currently two productivity tools comprise the PMT's: SVCS, a Software Version Control System and MAKE, an automated software generation tool. Together they control, examine, and automate the management of a software development project, greatly decreasing the time spent on tracking program changes and the generation of new systems.

Intel's Software Version Control System controls and documents software changes for both source and object files. SVCS handles storage and retrieval of different versions of a given module, controls update privileges, prevents different users from making changes independently, and requires all changes be throughly documented by recording who made what changes, when and why.

MAKE produces the specification of a 'minimum-work' job required to generate a new system. This job (i.e. submit file) typically includes compiles and links of the latest versions of specified source and object modules. If a newer source module exists for any specified object module, MAKE will specify a compile of this module, replacing the older module in the completed program. Unnecessary links and compiles, however, are eliminated. MAKE does the minimum work required to ensure consistent, up-to-date software, thus saving many hours of compiles and links.

This tutorial covers the operation and features of Intel's Program Management Tools. By carefully working step-by-step through the examples contained herein, the user should develop the requisite skills to fully exploit the many advantages PMT's provide. We strongly suggest the user work through all examples. Each example is carefully constructed to expose the new user to a wide variety of program features and use methodologies. A tutorial diskette, which is included in the PMT Software package, supplements this manual and greatly facilitates the learning process.

The user should completely read the User's Guide to Program Management Tools before using this tutorial, and be familiar programming in the ISIS environment. In addition, he or she should be using a Series III workstation operating under the ISIS-III operating system. Series II and IV users would need to adapt the command syntax in this tutorial for correct operation.

EXAMPLE OVERVIEW

This tutorial describes the design of a software system using the unique features of Intel's Program Management Tools. The example system, REMOTE, enables an iPDS™ development workstation to communicate with an NDS-II via an ISIS Cluster board. In this configuration the iPDS essentially functions as a dumb terminal. Two special commands, SEND and RECV, transfer files between the iPDS local storage and the NDS-II remote file system. A full discussion of REMOTE, including program listings, is covered in Appendix A.

The REMOTE program consists of five separate modules whose relationship is shown in Figure 1. The program is generated using the SUBMIT file in Figure 2. This tutorial shows how to design an SVCS database for REMOTE, and gradually alter the submit file into a MAKE file. In addition, variants of REMOTE are created to enable program execution on a Series II and also run under the CP/M operating system.

The tutorial diskette contains the many files described within this manual and simplifies the execution of some examples. Before using the diskette, the user's system must by configured with the following assignments:

```
:F0: - Directory containing ISIS
system files.

:F1: - Directory containing
tutorial source, object, executable
and CSD files. (All files supplied on
tutorial disk.)

:F2: - Directory containing 8 bit
compilers, linker, locater, MAKE and
SVCS.

:F3: - Directory containing 8 bit
library files, such as PLM80.LIB and
SYSTEM.LIB.

:F4: - Directory containing SVCS
database files created during
tutorial.

:F5: - Directory containing 16 bit
software, such as 16 bit SVCS and
MAKE.
```

In addition, the tutorial disk files COMMON.LIT and ISIS.EXT **must** be copied to the 8 bit system library directory (:F3:). After the above changes have been made, the tutorial files can be utilized without further modification.

PMT TUTORIAL

CONTENTS

PAGE

INTRODUCTION 1

EXAMPLE OVERVIEW 1

DATABASE IMPLEMENTATION USING SVCS 2

A. Database Design 2

B. Database Generation 3

USING MAKE 3

A. Generating the MAKE File 3

B. Adding SVCS to the MAKE File 4

CREATING VARIANTS USING SVCS .. 4

DATABASE OVERHEADS 7

A. Initial Set-Up 7

B. Adding Variants 7

C. Total Overhead 7

USING DEFAULTS TO SIMPLIFY OPERATIONS 7

OPERATION ON A STANDALONE SYSTEM 8

A. System Differences 8

B. MAKE File Changes 8

FIGURES 9

APPENDIX A A-1

USING SVCS

Database Design

SVCS manipulates UNITS which can have up to four parts or CLASSES. The allowable CLASS categories are:

- SOURCE, which contains the UNIT's source code;
- OBJECT, which contains the UNIT's object code;
- HISTORY, which contains the UNIT's history file;
- COMPOSITION, which can be used arbitrarily by the user.

Each UNIT is given a unique name and may utilize any or all of the above CLASSES.

Referring to Figure 1, the five main modules of our example program are REMOTE, RMSEND, RMRECV, FILEIO, and SERIAL. For each of these modules we will define an SVCS UNIT having the same name. In addition, for each UNIT we will create a SOURCE CLASS, which will contain the UNIT's PLM source code, a HISTORY CLASS, which will document changes to the source code, and an OBJECT CLASS, which will contain the UNIT's compiled source. The database is depicted in Table 1.

Table 1. Preliminary Database

UNIT NAME	SOURCE	HISTORY	OBJECT	COMPOSITION
REMOTE	REMOTE.PLM	As requested	REMOTE.OBJ	?
RMSEND	RMSEND.PLM	As requested	RMSEND.OBJ	?
RMRECV	RMRECV.PLM	As requested	RMRECV.OBJ	?
FILEIO	FILEIO.PLM	As requested	FILEIO.OBJ	?
SERIAL	SERIAL.PLM	As requested	SERIAL.OBJ	?

This simple database, however, does not contain all the subsystems required to generate the executable object code REMOTE. INCLUDE files, for example, are not included. Figure 3 shows all of the modules required to generate REMOTE. Additional units must be defined for the executable object code and all include files.

The executable code, REMOTE, is placed in new unit called EXEC. REMOTE will reside in EXEC's OBJECT class. In the SOURCE class we'll place REMOTE.MKE - the MAKE file (which we will create soon) that generates REMOTE. Finally, in the COMPOSITION class we will place the user's manual USER.MAN.

Include files are added to our database as new units and also as composition classes. The include files COMMON.LIT and ISIS.EXT, used by nearly all source files, are stored in two new units: COMMON and ISIS. FILEIO.EXT and SERIAL.EXT, however, are added to the database via the composition class of the units they're most closely associated with: FILEIO and SERIAL.

Two ISIS programs, SEND and RECV, are required for complete operation of the REMOTE program. These programs are added to the database as two new units called SEND and RECV. These units will store the source, object, and executable files for the two programs.

The database is now complete and shown in Table 2.

Table 2. Complete Database

UNIT NAME	SOURCE	HISTORY	OBJECT	COMPOSITION
REMOTE	REMOTE.PLM	As requested	REMOTE.OBJ	Not used
RMSEND	RMSEND.PLM	As requested	RMSEND.OBJ	Not used
RMRECV	RMRECV.PLM	As requested	RMRECV.OBJ	Not used
FILEIO	FILEIO.PLM	As requested	FILEIO.OBJ	FILEIO.EXT
SERIAL	SERIAL.PLM	As requested	SERIAL.OBJ	SERIAL.EXT
EXEC	REMOTE.MKE	As requested	REMOTE	USER.MAN
SEND	SEND.PLM	As requested	SEND.OBJ	SEND
RECV	RECV.PLM	As requested	RECV.OBJ	RECV
COMMON	COMMON.LIT	As requested	Not Used	Not Used
ISIS	ISIS.EXT	As requested	Not Used	Not Used

Every element in the database represents one portion of the REMOTE system diagram shown in Figure 3.

230879-001

Database Generation

Having designed the database it is a mechanical process to generate and initialize it. The steps required to do so are shown in Figure 4.

Referring to Figure 4, the first SVCS command‡:

```
 RUN :F5:SVCS.
ADMIN  :F4:REMOTE.DB  CREATE
```

creates the database master file. This file contains all the information relating to variants, unit names, default accesses and file protection. The name "REMOTE.DB" is the name of the database and will be used in all SVCS command references to the database.

The second command:

```
 ACCESS :F4:REMOTE.DB WR1 WW1
```

sets the access rights for the database. This must be done if multiple users access the database. SVCS propagates these access rights to all of its database files.

The next 28 commands create and fill the database units. Note how the SOURCE class of a unit may be filled when created by including the FROM option‡:

```
 RUN :F5:SVCS. ADMIN :F4:REMOTE.DB ADD
(UNIT = REMOTE FROM & :F1:REMOTE.PLM)
```

Also note how you must GET a COMPOSITION class, with write option, before you can PUT to it.

You may either submit MAKEDB.CSD (which is on the tutorial disk) or enter the commands one-by-one to generate this database. We suggest you enter the first few commands to become familiar with the SVCS command formats. Once comfortable with the commands, you can submit MAKEDB.CSD to finish the database construction. The completed database can be viewed in a structured printout generated by the following command‡:

```
 RUN :F5:SVCS.
ADMIN  :F4:REMOTE.DB  PRINT
```

USING MAKE

Generating the MAKE File

The submit file REMOTE.CSD, shown in Figure 2, will be used as the basis for generating the MAKE file. Ensure you understand the operation of REMOTE.CSD before continuing.

‡Command should be typed on one line.

Not shown in the submit file are source file dependencies on include files. These dependencies are hidden within the source files, and are displayed in Figure 3. Changing an include file is equivalent to changing a source file. For example, a change to COMMON.LIT modifies nearly all the source modules. MAKE and SVCS will be used to monitor and control include files too.

The files fall logically into three groups of executable files:

REMOTE – an executable file that runs on the remote system;
SEND – an executable file that runs on the ISIS-Cluster board to SEND a file to the Network file system;
RECV – an executable file that runs on the ISIS-Cluster board to RECEIVE a file from Network file system.

These groupings will be used within the MAKE file.

Figure 5 shows the first pass at creating a MAKE file. (Also refer to tutorial disk file REMMKE.EXM.) This MAKE file contains the least complicated MAKE constructs to keep the REMOTE project up to date.

Referring to Figure 5, the first MAKE command:

```
 $IF ALL > :f1:remote, send, recv THEN
 $END
```

is a trick used to fool MAKE into generating a dependency tree for three standalone or separate executable files. The next five commands test the time/date stamping of the five object files as compared to the source and associated include files. If the object file is older than the source or include files, then the source must be recompiled.

The last three MAKE commands test the age of the executable files REMOTE, SEND and RECV against the object modules that are linked to form these files. If any of the executable files are out of date, MAKE will add the appropriate task lines to the generated submit file to make them current.

Figure 6 shows the second pass in creating a MAKE file. (Also refer to tutorial disk file RMMKE1.EXM.) Note the macro definition:

```
 $SET work_device to ':F1:'
```

This definition substitutes the text ":F1:" for % work_device in all MAKE file commands. Macros enable the user to easily update MAKE files with future file changes.

The SET macro command may also specify a list of items which are used in a similar fashion with an iteration command. For example, the MAKE iteration command:

```
$FOR i IN %remote_files
```

will execute the FOR loop for each item defined in the SET macro of remote_files. The above iteration command converts the five MAKE commands in Figure 5 to one command in Figure 6.

Adding SVCS Constructs To The MAKE File

Figure 7 shows the final version of the MAKE file. (Also refer to tutorial disk file REMOTE.MKE.) SVCS has now been incorporated into this version, as well as the special macros %ALL, %TARGET, and %DEPEND. All code files (source and object) now refer to units within the REMOTE.DB SVCS database. For example, the following command retrieves the file :F1:SERIAL.EXT from the database:

```
 %get (serial,,cp) to
%"work_device"serial.ext
```

In addition, the special macros %ALL, %TARGET, and %DEPEND are used to simplify MAKE file coding.

We now add the final version MAKE file to the database with the command:

```
RUN :F5:SVCS. GET :F4:REMOTE.DB (EXEC)&
TO :BB: WRITE

RUN :F5:SVCS. PUT :F4:REMOTE.DB (EXEC)&
FROM :F1: REMOTE.MKE
```

After creating REMOTE.MKE, we run MAKE with the GENALL option to create a SUBMIT file that generates everything:

```
RUN :F5:MAKE. :F1:REMOTE.MKE TARGET&
(ALL) GENALL PRINT
```

The TARGET option ALL forces MAKE to produce a submit file that includes all task lines required to generate ALL (in this case REMOTE, SEND, and RECV). The default option is the first dependency node's target. The TARGET option overrides this default.

Finally, we can submit the MAKE file:

```
SUBMIT :F1:REMOTE
```

and generate the system.

CREATING VARIANTS USING SVCS

Before creating variants, let's review what we've done so far. In Section III we designed a SVCS database to store and control all program modules that comprise the software system REMOTE. We then constructed the database using the SVCS commands listed in Figure 4, which reside on the tutorial disk in the file MAKEDB.CSD. A MAKE file was then created in Section IV to automate system generation. Starting with the SUBMIT file in Figure 2, we created several versions of a MAKE file, each increasing in complexity. The final version contained substitution macros, enumeration macros, parameter macros, special macros such as %ALL, iteration commands, header and trailer commands, and SVCS constructs. The final MAKE file version is stored on the tutorial disk as REMOTE.MKE.

At this point the most challenging work has been completed. This section will cover variant creation and database management.

Let's save the WORK variant of the database to a saved variant, which we'll call ISISPDS.

We create this variant by issuing the command:

```
RUN :F5:SVCS. ADMIN  :F4:REMOTE.DB ADD&
(VARIANT = ISISPDS FROM WORK)
```

This command DOES NOT DOUBLE the size of the database! Many auxiliary files are generated, but they only contain pointers back into the original files.

We protect the variant ISISPDS by the command:

```
RUN :F5:SVCS. ADMIN  :F4:REMOTE.DB&
WRITEACCESS (ISISPDS = FALSE)&
DEFAULTACESS (ISISPDS = ALL)&
DEFAULTACCESS (WORK = BRIAN)
```

This command:

1. Sets the ISISPDS variant to read only.
2. Gives all users who access the database the known working ISISPDS variant.
3. Gives BRIAN, a system programmer, access to the WORK variant.

BRIAN is now the only person who can write to the database. In addition, when BRIAN checks out database files, he will get WORK variant files.

We will now modify files within our SVCS database and possibly the MAKE file to generate new versions of REMOTE that execute under ISIS Series II, CPMPDS, and CPM Series II.

We must incorporate the following changes:

- To change from ISISPDS to ISIS Series II execution **(creating the variant ISISS2):**

  Change the SERIAL.PLM file (Refer to tutorial disk file SERIAL.S2).

- To change from ISIS Series II to CPM Series II execution **(creating the variant CPMS2):**

  Change the REMOTE.MKE file (Refer to tutorial disk file REMMKE.CPM);

  Change the FILEIO.PLM file (Refer to disk file FILEIO.CPM).

- To change from CPM Series II to CPM PDS execution **(creating the variant CPMPDS):**

  Change the SERIAL.PLM file (Refer to disk file SERIAL.PDS)

  Note: All changes are made to the WORK variant. Then MAKE (using REMOTE.MKE) is run on WORK. After the WORK variant is updated for the new REMOTE program, WORK is moved to a new variant in the database. Upon completion of the following steps, five variants will be in the database: ISISPDS, ISISS2, CPMS2, CPMPDS and WORK. All variants will contain appropriate files including the executable REMOTE file. Remember, when a new variant is created by WORK, only modified files are added to the database. Unchanged files are not duplicated; instead pointers are set to the original files. (Section VI discusses database overhead.)

The following commands create the three new variants ISISS2, CPMS2 and CPMPDS.

1. Follow these steps to change and save a new variant called ISISS2 for holding the ISIS Series II version:

   a. First, give "ALL" access to the work variant, then get SERIAL.PLM out of the WORK variant and place it in the file SERIAL.PLM:

   ```
   RUN :F5:SVCS. ADMIN :F4:REMOTE.DB&
   DEFAULTACESS (WORK = ALL)

   RUN :F5:SVCS. GET :F4:REMOTE.DB&
   (SERIAL) TO :F1:SERIAL.PLM WRITE
   ```

   b. Make the necessary changes to the source code (see Appendix A for the changes) with the editor and then put the file back into the database: (Note: For the purpose of this tutorial, you can ignore the changes and just put the file back into the database.)

   ```
   RUN :F5:SVCS. PUT :F4:REMOTE.DB&
   (SERIAL) from :F1:SERIAL.PLM
   ```

   c. Run MAKE to examine all files that must be recompiled, relinked and relocated. Since the module dependencies have not changed, we can use the same MAKE file. Run MAKE by entering the commands.

   ```
   RUN :F5:SVCS. GET :F4:REMOTE.DB&
   (EXEC) to :F1:REMOTE.MKE

   RUN :F5: MAKE. :F1:REMOTE.MKE TARGET&
   (ALL) PRINT
   ```

   Note that the write option was not added in the first command. We do this because we don't want to change the file, just get it out of the database. If we now tried to PUT the file back into the database, SVCS would tell us the file was not checked out for writing and cannot be PUT back.

   d. Now submit the file REMOTE.CSD that MAKE built:

   ```
   SUBMIT :F1:REMOTE
   ```

   e. The database WORK variant now contains the ISIS Series II version of REMOTE. We save the WORK variant under another name by entering:

   ```
   RUN :F5:SVCS.    ADMIN :F4:REMOTE.DB&
   ADD (VARIANT = ISISS2 FROM WORK)
   ```

   Again, the database IS NOT COPIED, only header records and changed files are added. By creating this variant, we added a new version of SERIAL.PLM to the database. By now you should realize the efficiencies PMT's provide in managing a large software project.

   f. Now write protect the ISIS Series II version of REMOTE:

   ```
   RUN :F5:SVCS. ADMIN :F4:REMOTE.DB&
   WRITEACCESS(ISISS2=FALSE)&
   DEFAULTACCESS (ISISS2=BILL, FRANK)
   ```

If Bill or Frank now access the database (under default conditions) they will get the variant ISISS2. They may also specify a variant and get it. For example, if Bill wanted a copy of the unit (FILEIO) source from ISISPDS variant, he could issue the following command:

```
RUN :F5:SVCS. GET :F4:REMOTE.DB&
(FILEIO,WORK,SO) TO :F1:FILEIO.PLM
```

PMT's allow the database administrator to assign default variants to the appropriate people. This is extremely useful in complex multifunctional projects such as REMOTE.

2. The work variant contains the ISISS2 version of REMOTE. We can change and save a new variant called CPMS2 to hold the CPM Series II version by following these steps:

   a. First, get REMOTE.MKE and FILEIO:PLM (which must be changed) out of the WORK variant:

   ```
   RUN :F5:SVCS. GET :F4:REMOTE.DB&
   (FILEIO) TO :F1:FILEIO.PLM WRITE

   RUN :F5:SVCS. GET :F4:REMOTE.DB&
   (EXEC) TO :F1:REMOTE.MKE WRITE
   ```

   b. Now make the necessary changes to :F1:REMOTE.MKE and :F1:FILEIO.PLM (See Appendix A for the changes.) using an editor and then put them back in the database:

   ```
   RUN :F5:SVCS. PUT :F4:REMOTE.DB&
   (FILEIO) FROM :F1:FILEIO.PLM

   RUN :F5:SVCS. PUT :F4:REMOTE.DB&
   (REMOTE) FROM :F1:REMOTE.MKE
   ```

   c. Next, get and run the MAKE file:

   ```
   RUN :F5:SVCS. GET :F4:REMOTE.DB&
   (EXEC) TO :F1:REMOTE.MKE

   RUN :F5:MAKE. :F1:REMOTE.MKE TARGET&
   (ALL) PRINT
   ```

   d. And submit the submit file REMOTE.CSD:

   ```
   SUBMIT :F1:REMOTE
   ```

   e. The database WORK variant now contains the CPM Series II version of REMOTE. We save the WORK variant under the name CPMS2:

   ```
   RUN :F5:SVCS. ADMIN :F4:REMOTE.DB ADD&
   (VARIANT=CPMS2 FROM WORK)
   ```

   Here again, the database IS NOT COPIED, only header records and changed files are added. In creating this variant, we only added new copies of FILEIO.PLM and REMOTE.MKE to the database.

   f. Finally write protect CPMS2:

   ```
   RUN :F5:SVCS. ADMIN :F4:REMOTE.DB&
   WRITEACCESS (CPMS2=FALSE)&
   DEFAULTACCESS (CPMS2=NONE)
   ```

   NONE used as defaultaccess allows no one default access to the CPMS2 variant.

3. The WORK variant now contains the version that runs on the SERIES II under CPM. We change the WORK variant to the PDS CPM version by following these steps:

   a. Get SERIAL.PLM out of the WORK variant:

   ```
   RUN :F5:SVCS. GET :F4:REMOTE.DB&
   (SERIAL) TO :F1:SERIAL.PLM WRITE
   ```

   b. Make the necessary changes with the editor and then PUT the file back into the database:

   ```
   RUN :F5:SVCS. PUT :F4:REMOTE.DB&
   (SERIAL) FROM :F1:SERIAL.PLM
   ```

   c. Run MAKE.

   ```
   RUN :F5:SVCS. GET :F4:REMOTE.DB&
   (EXEC) TO :F1:REMOTE.MKE

   RUN :F5:MAKE. :F1:REMOTE.MKE TARGET&
   (ALL) PRINT
   ```

   d. And Submit:

   ```
   SUBMIT :F1:REMOTE
   ```

   e. The database WORK variant now contains the version of REMOTE that runs on the PDS under CPM. We save this version under another name:

   ```
   RUN :F5:SVCS. ADMIN :F4:REMOTE.DB&
   ADD (VARIANT=CPMPDS FROM WORK)
   ```

   f. Finally write protect CPMPDS by entering:

   ```
   RUN :F5:SVCS. ADMIN :F4:REMOTE.DB&
   WRITEACCESS (CPMPDS=FALSE)&
   DEFAULTACCESS (CPMPDS=NONE)
   ```

The database now has five variants: WORK, ISISPDS, ISISS2, CPMS2, and CPMPDS.

230879-001

DATABASE OVERHEAD

We will now examine the overhead SVCS added to the database.

Initial Set-Up

We put 23 different files into the database when it was originally set-up. (See Figure 4 for the CSD file used for set-up.) These files totaled 49,689 bytes. After they were put into the database, the database expanded to 51,204 bytes. 1,515 bytes were added to the files. This represents a 3% overhead putting the files into the SVCS database. Figure 8 shows a breakdown of this overhead.

Adding Variants

By adding a variant to the database without changing any files an overhead of 10 + (length of variant name) bytes is incurred. In this scenario, the database is not copied. Only pointers are added.

If you first change a file (or files) in an old variant and then copy the unit to the new variant, the overhead incurred is 10 + (length of variant name) bytes for each unchanged class plus 54 bytes for each changed file.

Note: The **changed** file is also **added** to the database, so the database **grows** by (file length) + 54 for each class change. However, the actual overhead incurred is only 54 bytes. The file changes would have to be stored on the disk even if we weren't using SVCS. Put another way, only files that are modified in a new version are copied. These files expand the database by 54 + (file length) bytes and expand the disk spaced used by the same amount. Without SVCS the additional disk space consumed would be (file length) bytes. The SVCS overhead is the difference between these two amounts, or 54 bytes.

Total Overhead

By adding the four variants to the database we added 691 bytes of overhead. Our overhead now totals: 2,206 bytes. The total size of the database (with all four executable files, sources, objects, etc.) is 88,025 bytes. Therefore, the overhead is only 2.5% of the database - a small price to pay considering all the features included with PMT's.

USING DEFAULTS TO SIMPLIFY OPERATION

While progressing through the tutorial, we have used the DEFAULTACCESS administration command several times. We will now discuss how the DEFAULTACCESS command can simplify the database administrator's job. First using the REMOTE example, let us set-up a hypothetical work environment.

Programmer	Variant working with
Brian	WORK
Bill, Frank	ISISPDS
John, Chris	ISISS2
Tim, Howard, Mary	CPMS2
Susan, Gordon	CPMPDS

Now, the command:

```
RUN :F5:SVCS. ADMIN :F4:REMOTE.DB&
DEFAULTACCESS (CPMS2 = Tim, Howard,&
Mary)
```

will set Tim's, Mary's and Howard's defaultaccess to CPMS2. (The names used are the NDS-II logon ID's, which SVCS uses to identify users.) If Tim entered the command:

```
RUN :F5:SVCS. GET :F4:REMOTE.DB&
(FILEIO) TO :F1:FILEIO.PLM
```

SVCS would get Tim's ID name from the system and use Tim's default variant, which is CPMS2. Tim would now have the CPMS2 version of FILEIO.PLM in his directory.

What if Tim wanted the FILEIO.PLM file in the ISISPDS variant? He would issue the following command and get it:

```
RUN :F5:SVCS. GET :F4:REMOTE.DB&
(FILEIO,ISISPDS) TO :F1:FILEIO.PLM
```

Note when the variant is specified; the default isn't checked.

The DEFAULTACCESS command aids the database administrator and system users by minimizing typing and by organizing which variants programmers are using.

Note: NONE and ALL may be used to set variants to the NONE or ALL default. However, if a user's default is changed, it is deleted from the previous list. Thus, if you set 15 people to 8 different variants and then set one variant to ALL, you will relinquish the old defaults and set everyone to the new one.

STANDALONE OPERATION

This section covers the use of PMT's on a Series III standalone system running ISIS II (W).

System Differences

Two major differences exist between standalone systems and networks: standalone systems do not have user ID's nor do they have date/time stamping.

SVCS must know which user is accessing the database in order to accurately check units and default accesses. Because standalone systems do not support user ID's, the user must add the string:

ID (user name)

to all commands.

MAKE uses the 'D' attribute on standalone systems to check file modification times. The 'D' attribute on ISIS II (W) marks when a file has been modified. When you edit a file called FILE1.PLM and save the changes, the file's 'D' attribute bit is set. This bit can only be reset with the following command:

```
ATTRIB FILE1.PLM DO
```

MAKE uses the 'D' attribute on standalone systems to tell if a file has been modified.

The next section lists the necessary MAKE file changes that must be made for MAKE operation on standalone systems.

MAKE File Changes

The command:

```
$IF FILE1.OBJ > FILE1.PLM, ISIS.EXT,
$COMMON.LIT THEN
     PLM80 FILE1.PLM
$END
```

is a typical MAKE construct for a MAKE file on a Network. On a standalone system, MAKE tests the file modification times (in the IF-THEN statement above) by testing if the 'D' attribute bit is set in any of the dependency files. If the bit is set, it assumes the files have been modified.

To use MAKE on a standalone system, we must add a command to the MAKE construct that resets the 'D' bit:

```
$IF FILE1.OBJ > FILE1.PLM, ISIS.EXT,
$COMMON.LIT THEN
     PLM80 FILE1.PLM
     ATTRIB %DEPEND DO
$END
```

The additional line in the above MAKE file resets the 'D' attribute of the dependant files when the generated SUBMIT file is run. These task lines will not be included in the SUBMIT file when MAKE is run again unless one of the dependency files is modified and its 'D' attribute is reset.

FIGURES

1. Remote Source Modules
2. SUBMIT File REMCSD.EXM
3. Remote System Diagram
4. SUBMIT File MAKEDB.CSD
5. MAKE File REMMKE.EXM
6. MAKE File RMMKE1.EXM
7. MAKE File REMOTE.MKE
8. Database Overhead

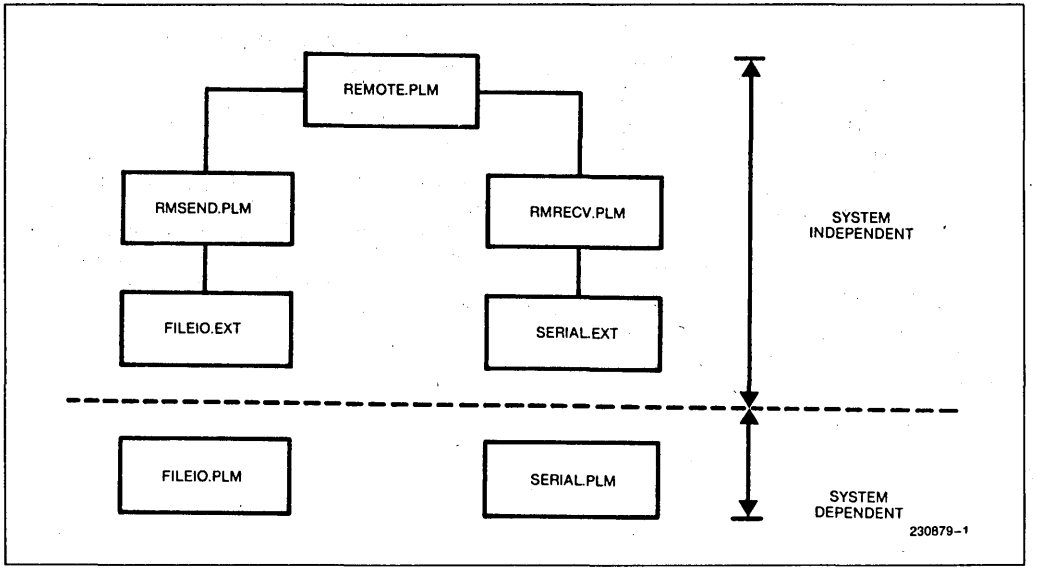


Figure 1. Shows the source modules used to create REMOTE.

REMOTE.PLM is the main module
RMSEND.PLM is a sub module used by REMOTE.PLM
RMRECV.PLM is a sub module used by REMOTE.PLM
FILEIO.PLM defines the operating system interfaces
FILEIO.EXT contains the external declarations of FILEIO.PLM
SERIAL.PLM defines the hardware interfaces of the SERIAL line
SERIAL.EXT contains the external declarations of SERIAL.PLM

It is expected that SERIAL.PLM and FILEIO.PLM will change for each system that REMOTE is configured for. The remaining modules are not expected to change.

Figure 2. Submit file used to generate REMOTE, RECV and SEND without using PMTs.

```
Page 1            :F1:REMCSD.EXM

; Submit file to generate the remote executable file that runs on the
; iPDS system under ISIS.
; Author :B. Valentine DSSO Applications Engineering 6/23/83

; Generate REMOTE file to run on the iPDS system

; First of all, compile all the source code.
:f2:plm80 :f1:remote.plm
:f2:plm80 :f1:rmsend.plm
:f2:plm80 :f1:rmrecv.plm
:f2:plm80 :f1:fileio.plm
:f2:plm80 :f1:serial.plm

; Now link them together
:f2:link :f1:remote.obj,:f1:rmsend.obj,:f1:rmrecv.obj,:f1:fileio.obj, &
:f1:serial.obj,:f3:plm80.lib,:f3:system.lib to :f1:remote.lnk

; Now locate the link file
:f2:locate :f1:remote.lnk symbols lines map print (:f1:remote.map)

; Move the file to the system directory
copy :f1:remote to remote b

; Generate SEND that runs on the network

:f2:plm80 :f1:send.plm
:f2:link :f1:send.obj,:f3:system.lib,:f3:plm80.lib to :f1:send.lnk
:f2:locate :f1:send.lnk
copy :f1:send to send b

; Generate RECV that runs on the network

:f2:plm80 :f1:recv.plm
:f2:link :f1:recv.obj,:f3:system.lib,:f3:plm80.lib to :f1:recv.lnk
:f2:locate :f1:recv.lnk
copy :f1:recv to recv b
```

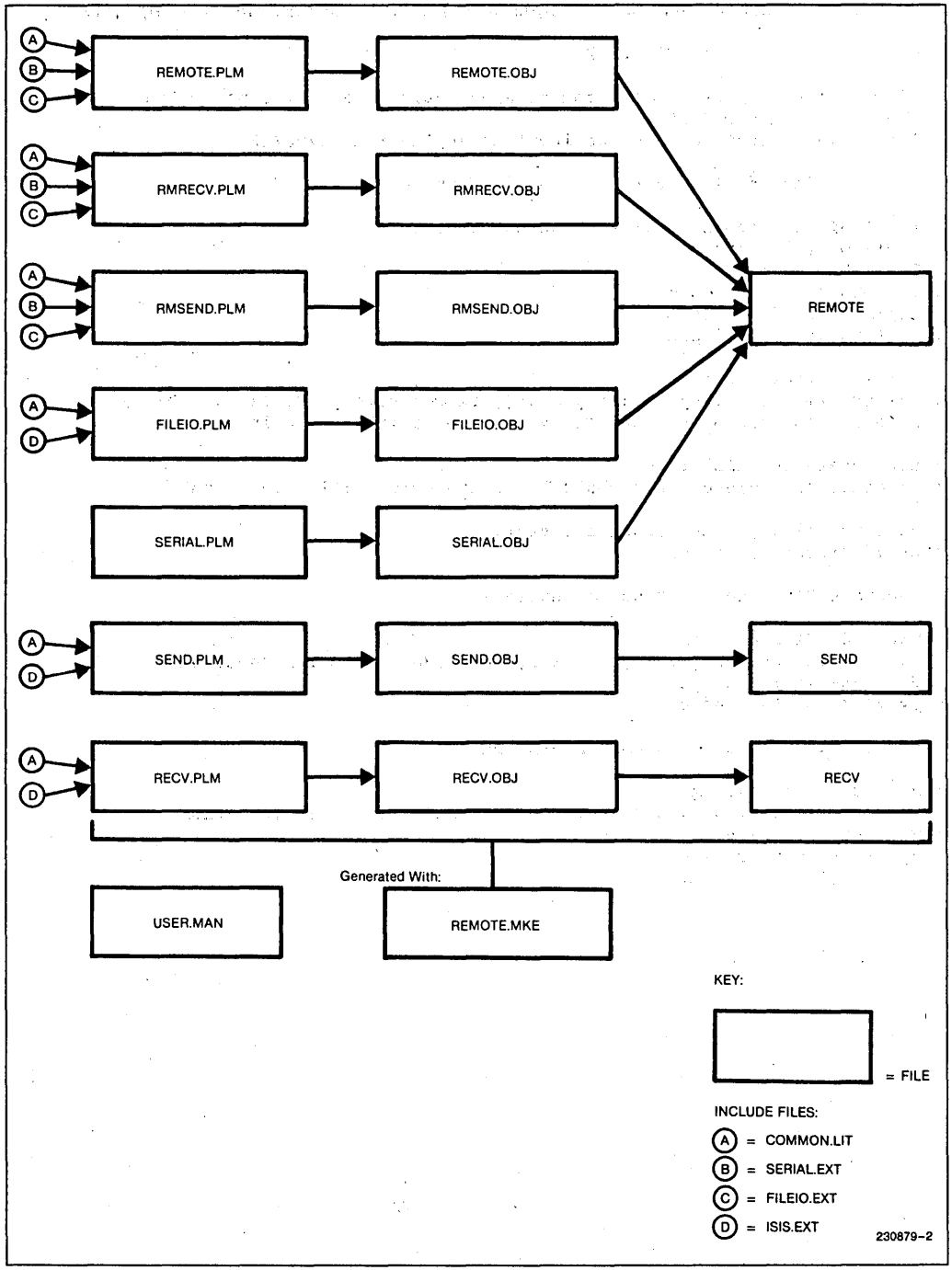


Figure 3. Remote System Diagram

230879-001

Figure 4. Submit file used to create and fill database. (Continued)

```
Page 2              :F1:MAKEDB.CSD

:f5:svcs. get    :f4:remote.db(fileio,,cp)          to :bb: write
:f5:svcs. put    :f4:remote.db(fileio,,cp)          from :fl:fileio,ext
:f5:svcs. admin :f4:remote.db add(unit = send    from :fl:send.plm)
:f5:svcs. put    :f4:remote.db(send,,oj)            from :fl:send.obj
:f5:svcs. get    :f4:remote.db(send,,cp)            to :bb: write
:f5:svcs. put    :f4:remote.db(send,,cp)            from :fl:send
:f5:svcs. admin :f4:remote.db add(unit = recv    from :fl:recv.plm)
:f5:svcs. put    :f4:remote.db(recv,,oj)            from :fl:recv.obj
:f5:svcs. get    :f4:remote.db(recv,,cp)            to :bb: write
:f5:svcs. put    :f4:remote.db(recv,,cp)            from :fl:recv
:f5:svcs. admin :f4:remote.db add(unit = common from :f3:common.lit)
:f5:svcs. admin :f4:remote.db add(unit = isis    from :f3:isis.ext)
:f5:svcs. admin :f4:remote.db add(unit = exec    from :fl:remote.mke)
:f5:svcs. put    :f4:remote.db(exec,,oj)            from :fl:remote
:f5:svcs. get    :f4:remote.db(exec,,cp)            to :bb: write
:f5:svcs. put    :f4:remote.db(exec,,cp)            from :fl:user.man

; Now the database is built, filled and saved in :f4:

; The following command creates a new variant called ISISPDS, by copying
; the WORK variant. In reality, no files are copied but pointers are set up
; pointing to the files for the ISISPDS variant; thus, disk space is conserved
; by using SVCS.

:f5:svcs. admin :f4:remote.db add(variant = isispds from work)

; Finally, the following command protects the database. Only BRIAN is given
; access to the WORK variant, everyone else is given access to database files
; in the ISISPDS variant. This SVCS feature permits the database administrator
; to assign variants to only those people needing the paticular versions.
; In addition, the writeaccess option sets the ISISPDS variant to read only;
; thus, no one can checkout the ISISPDS variant with write access.
:f5:svcs. admin :f4:remote.db writeaccess (isispds = false) &
    defaultaccess (isispds = all) defaultaccess (work = brian)

;
exit
```

230879-001

Figure 4. Submit file used to create and fill database.

```
Page 1          :F1:MAKEDB.CSD

; Submit file to create and fill the remote database from initial sources.
; It is submitted one time only. Once the database is created and filled,
; future changes are made using the SVCS ADD, PUT and GET commands.
; Device assignments are as follows:
;         :f0: - Directory containing ISIS system files
;         :f1: - Directory containing files supplied on tutorial disk
;         :f2: - Directory containing 8 bit system software files. Such
;                as PL/M-80, (8 bit) SVCS and MAKE compiler.
;         :f3: - Directory containing 8 bit libraries. Such as system.lib.
;                common.lit and isis.ext, supplied on the tutorial disk,
;                must be moved to this directory.
;         :f4: - Directory where all the database files will be created.
;         :f5: - Directory containing the 16 bit system software. Such as
;                PL/M-86, (16 bit) SVCS and MAKE.
; Author : B. Valentine - DSSO Applications Engineering 6/22/83

; Create the database
run :f5:svcs. admin :f4:remote.db create

; Since more than one person will access the database - make it shareable
; SVCS will propagate these access rights across all database files.
access :f4:remote.db wrl wwl

; Now that the database is created, fill it with the files supplied on the
; tutorial disk. The WORK variant will then contain the version of REMOTE
; for the iPDS running under ISIS..
;       ADD may be used to initialize SOURCE files, but
;       PUT must be used to initialize OBJECT,HISTORY or COMPOSITION files.
run

; Create a unit called remote and fill it with the source file remote.plm
:f5:svcs. admin :f4:remote.db add(unit = remote from :f1:remote.plm)

; Now fill the object class of the remote unit
:f5:svcs. put   :f4:remote.db(remote,,oj)       from :f1:remote.obj
; Note that if the variant name is not specified (remote,,oj), WORK is the
; default.

; Continue creating units and filling them with the files.
:f5:svcs. admin :f4:remote.db add(unit = rmsend from :f1:rmsend.plm)
:f5:svcs. put   :f4:remote.db(rmsend,,oj)       from :f1:rmsend.obj
:f5:svcs. admin :f4:remote.db add(unit = rmrecv from :f1:rmrecv.plm)
:f5:svcs. put   :f4:remote.db(rmrecv,,oj)       from :f1:rmrecv.obj
:f5:svcs. admin :f4:remote.db add(unit = serial from :f1:serial.plm)
:f5:svcs. put   :f4:remote.db(serial,,oj)       from :f1:serial.obj

; Note in the next command how you must GET (write permission) a composition
; unit before you can PUT to it. Since it has nothing in it yet, GET it to
; the byte bucket.
:f5:svcs. get   :f4:remote.db(serial,,cp)       to :bb: write
:f5:svcs. put   :f4:remote.db(serial,,cp)       from :f1:serial.ext

; Create and fill the remaining units of the database
:f5:svcs. admin :f4:remote.db add(unit = fileio from :f1:fileio.plm)
:f5:svcs. put   :f4:remote.db(fileio,,oj)       from :f1:fileio.obj
```

230879-001

Figure 5. First pass at building MAKE file. (See Figure 2 for submit file.)

```
Page 1            :F1:REMMKE.EXM

; Make file for the isis remote program
; This make program uses the least complicated make constructs to test
; the dependencies.
; Author : B. Valentine DSSO Applications Engineering 6/25/83

; Set the dependency tree for three separate executable files.
; Fool the MAKE utility into building the dependency tree for three
; unrelated (executable) programs by using ALL.
$IF all > :fI:remote, send, recv THEN
$END

; Note how in the next construct the source code include files are added.
; Also note how some of the files have the same right hand size dependencies
; accept for the changing of the file name. Pass 2 of the make file will
; show how these can be combined into an iteration loop.
$IF :f1:serial.obj > :f1:serial.plm THEN
    :f2:plm80 :f1:serial.plm
$END

$IF ;f1:fileio.obj > :f1:fileio.plm, :f3:isis.ext, :f3:common.lit THEN
    :f2:plm80 :f1:fileio.plm
$END

$IF :f1:remote.obj > :f1:remote.plm, :f3:common.lit, :f1:serial.ext,
$:f1:fileio.ext THEN
    :f2:plm80 :f1:remote.plm
$END

$IF :f1:rmsend.obj > :f1:rmsend.plm, :f3:common.lit, :f1:fileio.ext
$:f1:serial.ext THEN
    :f2:plm80 :f1:rmsend.plm
$END

$IF :f1:rmrecv.obj > :f1:rmrecv.plm, :f3:common.lit, :f1:fileio.ext,
$:f1:serial. ext THEN
    :f2:plm80 :f1:rmrecv.plm
$END

; Check the status of the remote executable file.
$IF :f1:remote > :f1:remote.obj, :f1:rmsend.obj, :f1:rmrecv.obj,
$:f1:fileio.obj, :f1:serial.obj THEN
    :f2:link :f1:remote.obj, :f1:rmsend.obj, :f1:rmrecv.obj, &
    :f1:fileio.obj, :f1:serial.obj, :f3:plm80.lib, :f3:system.lib to &
    :f1:remote.lnk
    :f2:locate :f1:remote.lnk symbols lines map print (:f1:remote.map)
$END

; Now that the remote program has been checked, check the two programs
; that run on the network.
; Check the NDS-II files RECV and SEND.!

; Check SEND
; Since there is only one module to the send program, we can test the
; executable file against the source code.
```

Figure 5. First pass at building MAKE file. (See Figure 2 for submit file.) (Continued)

```
Page 2              :F1:REMMKE.EXM

$IF send > :fl:send.plm, :f3:common.lit, :f3:isis.ext THEN
    :f2:plm80 :fl:send.plm
    :f2:link :fl:send.obj,:f3:system.lib,:f3:plm80.lib to :fl:send.lnk
    :f2:locate :fl:send.lnk
    copy :fl:send to send b
$END

; Check RECV
$IF recv > :fl:recv.plm, :f3:common.lit, :f3:isis.ext THEN
    :f2:plm80 :fl:recv.plm
    :f2:link :fl:recv.obj,:f3:system.lib, :f3:plm80.lib to :fl:recv.lnk
    :f2:locate :fl:recv.lnk
    copy :fl:recv to recv b
$END
```

Figure 6. Second pass of MAKE file. Note how macros and iteration are added.

```
Page 1              :F1:RMMKE1.EXM

; Second pass of the MAKE file for the REMOTE program.
; This pass has added the MAKE constructs of macros and iteration to
; pass one of the MAKE file.
; Author : B. Valentine DSSO Applications Engineering 6/25/83

; First of all define the macros for the MAKE file.
;  Define the substitution macros :
;    Substitution macros are used as constant defines. This way, if
;    a major change is made, such as the source code device changes
;    from :fl: to :f2:, the only update to the MAKE files is to change
;    the macro definition

$    SET work_device      to ':fl:'
$    SET 8_bit_exe        to ':f2:'
$    SET 8_bit_lib        to ':f3:'
; Note how macros may be nested and the macro is used with the %"<name>".
$     SET plm             to '%"8_bit_exe"plm80'
$     SET locate          to '%"8_bit_exe" locate'
$     SET link            to '%"8_bit_exe"link'
$     SET syslib          to '%"8_bit_lib"system.lib'
$     SET plmlib          to '%"8_bit_lib"plm80.lib'
$     SET comlit          to '%"8_bit_lib"common.lit'
$     SET isis            to '%"8_bit_lib"isis.ext'
; Now define the enumeration macros :
$     SET nds2_files      to (recv,send)
$     SET remote_files    to (rmrecv,rmsend)
; Now start the dependecies
```

230879-001

Figure 6. Second pass of MAKE file. Note how macros and iteration are added. (Continued)

```
; Set the dependency tree for three separate executable files.
$IF all > %"work_device"remote, send, recv THEN
$END

$IF %"work_device"serial.obj > %"work_device"serial.plm THEN
    %plm %"work_device"serial.plm
$END

$IF %"work_device"fileio.obj > %"work_device"fileio.plm, %comlit, %isis THEN
    %plm %"work_device"fileio.plm
$END

$IF %"work_device"remote.obj > %"work_device"remote.plm,
$%comlit, %"work_device"serial.ext, %work_device"fileio.ext THEN
    %plm %"work_device"remote.plm
$END

$FOR i IN %remote_files
; Build the send and receive modules for the remote system.
$    IF %work_device%"i".obj > %work_device%"i".plm, %comlit,
$    %"work_device"fileio.ext, %"work_device"serial.ext THEN
       %plm %work_device%"i".plm
$    END
$END

; Check the remote executable file
$IF %"work_device"remote > %"work_device"remote.obj,
$%"work_device"rmsend.obj, %"work_device"rmrecv.obj,
$%"work_device"fileio.obj, %"work_device"serial.obj,
$%plmlib, %syslib THEN
    %link %"work_device"remote.obj, &
      %"work_device"rmsend.obj, %"work_device"rmrecv.obj, &
      %"work_device"fileio.obj, %"work_device"serial.obj, &
      %plmlib, %syslib to %"work_device"remote.lnk
    %locate %"work_device"remote.lnk symbols lines &
      map print (%"work_device"remote.map)
$END

; Now that the remote program has been checked, check the two programs
; that run on the network.
$FOR i IN %nds2_files
; Check the NDS_II files RECV and SEND.
$    IF %i > %work _device%"i".plm, %comlit, %isis THEN
        %plm %work_device%"i".plm
        %link %work_device%"i".obj,%syslib, %plmlib to %work_device%"i".lnk
        %locate %work_device%"i".lnk
        copy %work_device%"i" to %i b
$   END
$END
```

230879-001

Figure 7. Final pass of MAKE file

```
Page 1          :F1:REMOTE.MKE

; MAKE file for the isis REMOTE program that runs on the iPDS system.
; Author : B. Valentine DSSO Applications Engineering 6/25/83

; First of all define the macros for the MAKE file.
;  Define the substitution macros:
;    Substitution macros are used as constant defines. This way, if
;    a major change is made, such as the source code device changes
;    from :f1: to :f2:, the only update to the MAKE file is to change
;    the macro define.

$    SET work_device       to ':f1:'
$    SET 8_bit_exe         to ':f2:'
$    SET 8_bit_lib         to ':f3:'
$    SET database          to ':f4:remote.db'
$    SET svcs_drive        to 'run :f5:'

$    SET plm               to '%"8_bit_exe"plm80'
; Note how macros may be nested and the macro is used with the %"<name>.
$    SET locate            to '%"8_bit_exe"locate'
$    SET link              to '%"8_bit_exe"link'
$    SET syslib            to '%"8_bit_lib"system.lib'
$    SET plmlib            to '%"8_bit_lib"plm80.lib'
$    SET comlit            to '%"8_bit_lib"common.lit'
$    SET get               to '%"svcs_drive"svcs_get %database'
$    SET put               to '%"svcs_drive"svcs_put %database'

; Now define the enumeration macros:
$    SET nds2_files        to (recv,send)
$    SET remote_files      to (remote,fileio,serial,rmrecv,rmsend)
$    SET files             to (%all(%nds2_files),%all(%remote_files))

; Tell make that we are going to be looking at the files in the database.
$ FOR i in %files
$    svcs %work_device%"i".plm   =%database (%i)
$    svcs %work_device%"i".obj   =%database (%i,,oj)
$END
$ svcs %"work_device"serial.ext  =%database (serial,,cp)
$ svcs %"work_device"fileio.ext  =%database (fileio,,cp)
$ svcs %"work_device"remote      =%database (exec,,oj)
$ svcs %"work_device"send        =%database (send,,cp)
$ svcs %"work_device"recv        =%database (recv,,cp)

; The include files are always required, so get them with the header.
$ HEADER
; Get all the externals and include files from the database
    %get (serial,,cp)     to %"work_device"serial.ext
    %get (fileio,,cp)     to %"work_device"fileio.ext
$END

; Now start the dependecies

;Set the dependency tree for three separate executable files.
;IF all > %"work_device"remote, %all(%work_device%nds2_files) THEN
$END
```

230879-001

Figure 7. Final pass of MAKE file (Continued)

```
Page 2              :F1:REMOTE.MKE

$FOR i IN %remote_files
; Build all the object files in the remote program.
$     IF %work_device%"i".obj > %work_device%"i".plm, %comlit,
$     %"work_device"fileio.ext, %"work_device"serial.ext THEN
         %get (%i) to %work_device%"i".plm
         %plm %work_device%"i".plm
         %put (%i,,oj) from %target
$     END
$END

; Check the remote executable file that runs on the iPDS system.
$ IF %"work_device"remote > %all(%work_device%"remote_files".obj),
$ %plmlib, %syslib THEN
$     FOR i in %remote_files
         %get (%i,,oj) to %work_device%"i".obj
$     END
     %link %depend to %"work_device"remote.lnk
     %locate %"work_device"remote.lnk symbols lines &
          map print(%"work_device"remote.map)
     %put (exec,,oj) from %target
$ END

; Now that the remote program has been checked, check the two programs
; that run on the network.


$FOR i IN %nds2_files.
; Check the NDS_II files RECV and SEND.
$     IF %work_device%i > %work_device%"i".plm THEN
         %get (%i) to %depend
         %plm %depend
         %put (%i,,oj) from %work_device%"i".obj
         %link %work_device%"i".obj, %syslib, %plmlib to %work_device%"i".lnk
         %locate %work_device%"i".ink
         %get (%i,,cp) to :bb:write
         %put (%i,,cp) from %target
$     END
$END
```

Figure 8 Breakdown of initial overhead of database.

Note: All numbers are in bytes

Byte Length		
266	Database header file	
62	Creating a unit in the database	
95	**Filling classes of a unit**	(Maximum)
	First Class	41
	Second Class	18
	Third Class	18
	Fourth Class	18
	TOTAL	95

Note: Class overhead is calculated for only the classes filled. Example, If a unit has only two classes filled, the overhead is 41 + 18 = 59.

In REMOTE, there are 10 units.
The overhead breaks down as follows:

Byte Length		
266	Database header file	
620	Unit header files 10 units × 62	
629	**Unit Classes**	
	2 units with 1 class filled 2×41	=82
	3 units with 2 classes filled 3×(41+18)	=162
	5 units with 3 classes filled 5×(41+18+18)	=385
	TOTAL	629
1,515	GRAND TOTAL	

230879-001

APPENDIX A

CONTENTS

PAGE

USERS MANUAL A-2
USER. MAN A-2

INCLUDE FILES A-3
FILEIO.EXT A-3
COMMON.LIT A-4
SERIAL.EXT A-4
ISIS.EXT A-5

REMOTE SOURCE FOR ISIS, iPDS ... A-7
REMOTE.PLM A-7
RMRECV.PLM A-10
RMSEND.PLM A-13
FILEIO.PLM A-15
SERIAL.PLM A-17

SOURCE CODE FOR RECV AND SEND NETWORK FILES A-17
RECV.PLM A-17
SEND.PLM A-19

CHANGES TO SERIAL.PLM TO RUN ON A SERIES III A-21
SERIAL.S2 ..... A-21

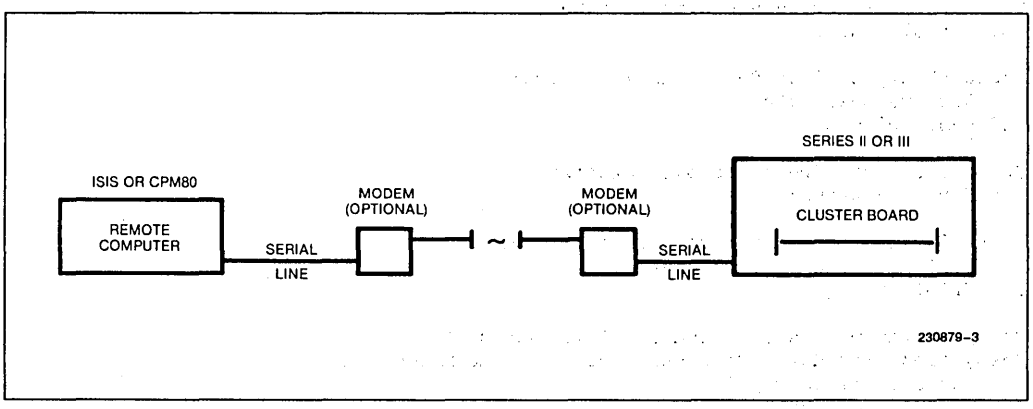
CHANGES TO FILEIO.PLM AND REMOTE.MKE TO RUN UNDER CPM A-22
FILEIO.CPM A-22
REMOTE.CPM A-26

USERS MANUAL

```
:F1:USER.MAN
```

This is the users manual for the PMT tutorial Remote program.

REMOTE is a program that runs on a remote computer connected to an NDS-II via an ISIS cluster board. The connection is an RS232 line and may include modems as shown below.



REMOTE, for the most part, turns the remote computer into a dumb terminal. ie. Characters entered on the remote computer keyboard are sent down the serial line and characters received up are echoed to the remote computers screen.

REMOTE internally collects the characters typed by the remote computer user and saves them in a buffer. When a <CR> is entered, REMOTE scans the buffer looking for three special commands - SEND, RECV and LOCAL. If these commands are not found - operation as a dumb terminal continues.

If one of these special commands are intercepted, REMOTE flips into file transfer mode (SEND or RECV commands) or back to standalone operation. Also, SEND or RECV cause a program to be activated on the cluster board to effect the file transfer. A simple protocol is used (STX, 128 data bytes, CHECKSUM, ETX) so some error checking is done.

REMOTE is written to be configurable. The I/O system is defined in FILEIO.PLM and the serial line configuration is SERIAL.PLM. All of the software is written in PLM. The systems currently supported are:

```
FILEIO.PLM - ISIS and CPM80

SERIAL.PLM - Series-II and iPDS
```

Thus four variants are currently available.

Note: Device 0 on the network must contain the SEND and RECV executable files.

INCLUDE FILES

```
:F1:FILEIO.EXT

/*     In case of fatal errors */
Exit: Procedure external;
end Exit;

/*    Operating system dependant Console Routines */
Console$Input: Procedure byte external;
end Console$Input;

Console$Output: Procedure (char) external;
    Declare char byte;
end Console$Output;

Console$Status: Procedure byte external;
end Console$Status;

Print$String: Procedure (string$ptr) external;
    Declare string$ptr pointer;
end Print$String;

/*    Operating system dependant file routines */
Open$file: Procedure (file$ptr,mode) byte external;
    Declare file$ptr pointer,
            mode byte;
end Open$File;

Create$file: Procedure (file$ptr) byte external;
    Declare file$ptr pointer;
end Create$File;

Read$sector: Procedure (file$id, buffer$ptr) byte external;
    Declare file$id byte,
            buffer$ptr pointer;
end Read$sector;

Write$sector: Procedure (file$id, buffer$ptr) byte external;
    Declare file$id byte,
            buffer$ptr pointer;
end Write$sector;

Close$file: Procedure (file$id) byte external;
    Declare file$id byte;
end Close$File;

$list
```

```
:F1:COMMON.LIT

/* Some useful defines for the remote program */

declare lit        literally 'literally';
declare word       lit       'address';
declare pointer    lit       'address';
declare connection lit       'address';

declare cr     lit '0dh',
        lf     lit '0ah',
        TAB    lit '09h',
        SOH    lit '01h',
        STX    lit '02h',
        ETX    lit '03h',
        EOT    lit '04h',
        ACK    lit '06h',
        NAK    lit '15h',
        XON    lit '11h',
        XOF    lit '13h',
        CAN    lit '18h',
        SUB    lit '1ah',
     RUBOUT    lit '7fh';

declare forever lit 'while 1';

declare false    lit '0',
        true     lit 'not false';

declare read$only    lit '1',
        write$only   lit '2',
        read$write   lit '3';

$list



:F1:SERIAL.EXT

/* Front end externals for the serial.plm link to the remote logon */
Serial$Status: Procedure byte external;
end Serial$Status;

Serial$Input: Procedure byte external;
end Serial$Input;

Serial$Output: Procedure (char) external;
    Declare char byte;
end Serial$Output;

Serial$Control: Procedure (value) external;
    Declare value byte;
end Serial$Control;

$list
```

230879-001

```
:F1:ISIS.EXT

isis: procedure (type, parameter$ptr) external;
    declare type byte,
            parameter$ptr address;
end isis;

open: procedure (conn$p, path$p, access, echo, status$p) external;
    declare (conn$p, path$p, access, echo, status$p) address;
end open:

close: procedure (conn, status$p) external;
    declare (conn, status$p) address;
end close;

read: procedure (conn, buff$p, count, actual$p, status$p) external;
    declare (conn, buff$p, count, actual$p, status$p) address;
end read;

write: procedure (conn, buff$p, count, status$p) external;
    declare (conn, buff$p, count, status$p) address;
end write;

seek: procedure (conn, mode, block$p, byte$p, status$p) external;
    declare (conn, mode, block$p, byte$p, status$p) address;
end seek;

rescan: procedure (con, status$p) external;
    declare (conn, status$p) address;
end rescan;

spath: procedure (path$p, info$p, status$p) external;
    declare (path$p, info$p, status$p) address;
end spath;

delete: procedure (path$p, status$p) external;
    declare (path$p, status$p) address;
end delete;

rename: procedure (old$p, new$p, status$p) external;
    declare (old$p, new$p, status$p) address;
end rename;

attrib: procedure (path$p, attrib, on$off, status$p) external;
    declare (path$p, attrib, on$off, status$p) address;
end attrib;

consol: procedure (ci$p, co$p, status$p) external;
    declare (ci$p, co$p, status$p) address;
end consol;

load: procedure (path$p, load$offset, switch, entry$p, status$p) external;
    declare (path$p, load$offset, switch, entry$p, status$p) address;
end load;

whocon: procedure (conn, buff$p) external;
    declare (conn, buff$p) address;
```

230879-001

```
:F1:ISIS.EXT

end whocon;

error: procedure (error$num) external;
    declare (error$num) address;
end error;

de$time: procedure (dt$p, status$p) external;
    declare (dt$p, status$p) address;
end de$time;

filinf: procedure (file$table$p, mode, file$info$p, status$p) external;
    declare (file$table$p, file$info$p, status$p) address,
            mode byte;
end filinf;

getd: procedure (did, conn$p, count, actual$p, table$p, status$p) external;
    declare (did, conn$p, count, actual$p, table$p, status$p) address;
end getd;

exit: procedure external;
end exit;

ci: procedure byte external;
end ci;

co: procedure (char) external;
    declare (char) byte;
end co;

ri: procedure byte external;
end ri;

po: procedure (char) external;
    declare (char) byte;
end po;

lo: procedure (char) external;
    declare (char) byte;
end lo;

csts: procedure byte external;
end csts;

iodef: procedure (type, entry) external;
    declare type byte,
            entry address;
end iodef;

iochk: procedure byte external;
end iochk;

ioset: procedure (value) external;
    declare value byte;
end ioset;

memck: procedure address external;
end memck;

$list
```

230879-001

REMOTE SOURCE FOR ISIS, iPDS

```
:F1:REMOTE.PLM

$DEBUG
Remote$Logan: do;

/*
    This program runs on a remote computer connected via a serial line to
    an ISIS Cluster board on an NDS-II system.
    The remote computer may be connected to the ISIS cluster board via a modem.
    It enables the remote computer to use all of the facilities of the NDS-II.
    For the most part the remote computer behaves as a dumb terminal; two
    commands (SEND and RECV) are intercepted to enable file tranfer
    between the remote computers file system and the NDS-II file system.
    Most remote computers will not be able to keep up with a high speed serial
    line so a XON/XOF protocol is used to slow the serial line down if required.
    Author: B. Valentine 6/22/83 - DSSO Applications Engineering
*/

$nolist include (:f3:common.lit)
$include(:f1:serial.ext)
$nolist include(:f1:fileio.ext)

Declare buffer$ptr              byte public;
Declare buffer(128)             byte public;
Declare save$buffer(128)        byte;
Declare save$buffer$ptr         byte;
Declare character               byte;
Declare time$out                byte;
Declare saved                   byte;
Declare i                       byte;

Send: Procedure external;
end Send;

Receive: Procedure external;
end Receive;

Uppercase: Procedure (char) byte;
/*If the character passed in is lowercase then convert it to uppercase.
*/
Declare char byte;
    if ((char>= 'a') AND (char <= 'z')) the return (char - 20h);
    return char;
end Uppercase;

Put$in$buffer: Procedure (character):
/* Put the character passed in into the input buffer, checking for EOLN
   and rubout.
*/
Declare character byte;
    character = uppercase (character);
    if character = RUBOUT then do;
        if buffer$ptr <> 0 then buffer$ptr = buffer$ptr - 1;
        return;
```

230879-001

```
:F1:REMOTE.PLM

    end;
    if character = cr then buffer$ptr = 0;
    else if character = lf then do;
        /* Mark end of the buffer */
        buffer (buffer$ptr) = ' ';
        buffer$ptr = buffer$ptr + 1;
        buffer(buffer$ptr) = lf;
        return;
    end;
    else if character >= ' ' then do;
        buffer(buffer$ptr) = character;
        buffer$ptr = buffer$ptr + 1;
    end;
end Put$in$buffer;

Match$Keyword: Procedure byte;
/* Check command to see if it's one to process or just send down to the
   Network.
*/
Declare Keywords (4) structure (text(7) byte) data
    ('SEND   ',
     'RECV   ',
     'LOGOFF ',
     'LOCAL  ');
Declare (index,match,i) byte;
    do index = 0 to 3;
        match = true;
        do i = 0 to 6;
            if (Keywords(index).text (i) = ' ') and (match) then return index;
            if buffer(i) <> Keywords(index).text(i) then match = false;
        end;
    end;
return 4; /* No match */
end Match$Keyboard;

/************************* Program starts here ************************/
buffer$ptr = 0;
do i = 0 to 127;
    buffer(i) = ' ';
end;

/* Say Hello to the user */
call Print$String(.(cr,lf,
'REMOTE LOGON TO NDS 2. X1.8',cr,lf,
'--------------------',cr,lf,lf,
'Trying to establish connection......',cr,lf,'$'));

/* Kick start the serial line */
/* Send a BREAK */
call Serial$Control(00111111b);
call time(200);
call Serial$Control(00110111b);

call Serial$Output(cr);
```

230879-001

```
:F1:REMOTE.PLM

call Serial$Output(cr);

/* Set up as a transparent terminal */
do forever;
    if Console$Status then do;
        character = Console$Input and 7FH;
/* Need to scan for SEND and RECV commands */
        if character = cr then do;
            call put$in$buffer(lf);   /* Mark end of buffer */
            do case match$keyword;
                call send;
                call receive;
                do;             /* User typed Logoff, so do it */
                    call Serial$Output(cr);
                    call Exit;
                    end;
                do;
                    call Serial$Output(CAN);
                    call Exit;
                    end;
                ; /* Do nothing */
                end;
            do i = 0 to 127;    /* Clear buffer after comparison */
                buffer(i) = ' ';
                end;
            end;
        call put$in$buffer(character);
        call Serial$Output(character);
        end;

    if Serial$Status then do;
        character = Serial$Input and 7FH;

/* We need time to deal with a LF */
        if character = lf then do;
            call Serial$Output(X0F);
            call Serial$Output(X0F);
/* Stop characters being sent to me, note that a few will be on the way...
Collect them...
*/
            save$buffer(0) = lf;
            save$buffer$ptr = 1;
            do time$out = 0 to 100;
               call time(2); /* 100 microseconds */
               if Serial$Status then do;
                    save$buffer(save$buffer$ptr) = Seral$Input;
                    save$buffer$ptr = save$buffer$ptr + 1;
                    time$out = 0; /* Reset the timeout */
                    end;
               end;
/* Get here once no characters are waiting. Send saved characters to screen */
            do saved = 0 to save$buffer$ptr - 1;
               call Console$Output(save$buffer(saved));
               end;
/* We have now caught up so.... */
            call Serial$Output(X0N);

            end;
        ELSE call console$output(character);
        end;
    end; /* Do forever */
end Remote$Logon;
```

230879-001

```
:F1:RMRECV.PLM

$DEBUG
Receive: do;

/*    This is part of the REMOTE_LOGON program. This module is called if the
      remote computer is to receive a file from the Network.
*/

/*    Declare the variables used from Remote$Logon */
Declare buffer$ptr    byte external;
Declare buffer(128)    byte external;
Declare file$id byte external;

$nolist include(:f3:common.lit)
$nolist include(:f1:fileio.ext)
$nolist include(:f1:serial.ext)

Declare delimit(4) byte data ('FROM');

Declare (character, i, j, match, status) byte;
Declare (count, checksum, received$checksum, loop$count, end$buffer) byte;

Wait$for$serial$input: Procedure (no$time$limit) byte;
Declare (no$time$limit, character, time$out) byte;
    do time$out = 0 to 100;
/* Has the user aborted the command? */
        if Console$Status then do;
            character = Console$Input and 7FH;
            if character = CAN then call Exit;
        end;
        if Serial$Status then return Serial$Input;
        call time(2); /* 100 microseconds */
        if no$time$limit then timeout = 0;
    end;
    call Print$String(.(cr,lf,lf,'Serial line lost, Program aborted',
                        cr,lf,'$'));
    call Exit;
end Wait$for$serial$input;

buf$all$blanks: Procedure (Buf$ptr) byte;
/* Check to see if the remainder of the buffer is blanks. */
Declare buf$ptr address,
        (buf based buf$ptr) (1) byte,
        i byte;
    i = 0;
    do while (buf(i) <> lf) and (buf(i) = ' ');
        i = i + 1;
    end;
    if buf(i) = lf then return true;
    return false;
end buf$all$blanks;

Receive: Procedure public;
/*    Copy a file from the NDS-II to the Remote Computer */
```

```
:F1:RMRECV.PLM

/*    Say HELLO */
  call Print$String(.(cr,lf,'Receive V2.3',cr,lf,'$'));

/*    Skip through the command line looking for Remote Computer filename */
    loop$count = 0;
    end$buffer = 0;
    /* Find the end of good characters in the buffer */
    do while buffer(end$buffer) <> lf;
        end$buffer = end$buffer + 1;
    end;
    if end$buffer < 14 then do;
        /* Not enough characters in buffer to even get started. Must have
           at least "RECV A FROM B<lf>", which is 14 characters.
        */
        call Print$String(.('Command syntax error. Correct format is :',cr,lf,
            'RECV <remote_file> FROM <NDS_II_file>',cr,lf,'$'));
        call Serial$Output(CAN);
        return;
    end;
    i = 5;   /* Skip over the recv command word */
    if buf$all$blanks(.buffer(i)) then do;
        /* Buffer passed the length requirement but is all blanks after
           the RECV command word.
        */
        call print$String(.('Command syntax error. Correct syntax is:',cr,lf,
            'RECV <remote_file> FROM <NDS_II_file>',cr,lf,'$'));
        call Serial$Output(CAN);
        return;
    end;

    do while buffer(i) = ' ';        /* Skip blanks before local file name */
        i = i +1;
    end;

/*    We have found the filename */
/*    Check if it exists */

    file$id = Create$File(.buffer(i));
    if file$id = 0ffh then do;
        call Print$String(.('Local File already exists',cr,lf,'$'));
        call Serial$Output(CAN);    /* Abort ISIS command */
        return;
    end;

/* Now that the file is good - see if FROM is in the command string */

do while buffer(i) <> ' ';     /* Skip the local file name */
        i = i + 1;
    end;
        do while buffer(i) = ' '; /* Skip blanks before <FROM> */
        i = i + 1;
    end;
    match = true;
    do j = 0 to 3;            /* Check for <FROM> in command string */
        if buffer (i+j) <> delimit(j) then match = false;
```

230879-001

:F1:RMRECV.PLM

```
    end;
    if match then do;
/*    File OK, activate the ISIS RECV command */
      call Serial$Output(cr);

/*    Skip passed CR,LF sent from ISIS. If no problems at ISIS end of the link
      then we will be sent a STX.
*/
      character = Wait$for$serial$input(true); /* CR */
      character = Wait$for$serial$input(true); /* LF */
      do forever;
Try$Again:  character = Wait$for$serial$input(true); /* STX? */
            if character = ETX then do;
                status = Close$File(file$id);
                if status = 0ffh then call Print$String(.(
                    'Local disk write-protected',cr,lf,'$'));
                else call Print$String(.(
                    cr,lf,'File transfer complete',cr,lf,'$'));
                return;
            end;
            if character <> STX then do;
                 return;
            end;

/*    ISIS is about to send us a buffer */
            checksum = 0;
            do i = 0 to 127;
                character = Wait$for$serial$input(false);
                buffer(i) = character;
                checksum = checksum + character;
            end;
            received$checksum = Wait$for$serial$input(false);
            character = Wait$for$serial$input(false); /* ETX */
            if checksum <> received$checksum then do;
                /* Checksum error - request retransmission */
                call Console$Output('?');
                call Serial$Output (NAK);
                goto Try$Again;
            end;
/*    Buffer received OK */
/*    Write to disk */
            call Console$Output(loop$count + '0');
            loop$count = (loop$count + 1) MOD 10;
            status = Write$Sector(file$id, .buffer);
            if status <> 0 then do;
                status = Close$File(file$id);
                call Print$String(.('Local disk full',cr,'$'));
                return;
            end;
/*   Buffer written to disk, Look for some more..... */
            call Seral$Output(ACK);
       end; /* Do forever */

/*    String did not match, keep looking */
```

230879-001

```
:F1:RMRECV.PLM

    end;

/*    Have now scanned the complete command line */
    call Print$String(.('Missing <FROM>. Correct syntax is:',cr,lf,lf,
                'RECV <LOCAL_FILENAME> FROM <NDS_II_FILENAME>',cr,lf, '$'));
/*    Abort ISIS command too */
    call Serial$Output(CAN);
    end Receive;
end Receive;


Page 1            :F1:RMSEND.PLM

$DEBUG
Send: do;

/*    This is part of the REMOTE-LOGON program. This module called if
      user is going to send a file from the remote computer to the
      NDS-II.
*/

/*    Declare the variables used from Remote$Logon */
Declare buffer$ptr    byte external;
Declare buffer(128)    byte external;
Declare file$id byte public;

$nolist include(:f3:common.lit)
$nolist include(:f1:fileio.ext)
$nolist include(:f1:serial.ext)

Declare delimit(4) byte data (' TO ');

Declare (character, i, match, status, count) byte;
Declare (checksum, received$checksum, loop$count) byte;

Send: Procedure public;
/*    Copy a file from the Remote Computer to the NDS 2 */

/*    Say HELLO */
    call Print$String(.(cr,lf,'Send V2.1',cr,lf,'$'));

/*    Skip through the command line looking for Remote Computer filename */
    loop$count = 0;
    do i = 0 to buffer$ptr;
        if buffer(i) = ' ' then do;
            do while buffer(i) = ' ';
                i = i + 1;
                end;
            file$id = Open$File(.buffer(i),read$only);
            if file$id = 0ffh then do;
                call Print$String(.('Local file does not exist',cr,lf,'$'));
                call Serial$Output(CAN):    /* Abort ISIS command */
                return;
                end;
/*    File OK, activate the ISIS SEND command */
            call Serial$Output (cr);
```

230879-001

```
:F1:RMSEND.PLM

/*    Skip passed CR,LF sent from ISIS. If no problems at ISIS end of the link
      then we will be sent a ACK.
*/
            character = Serial$Input; /* CR */
            character = Serial$Input; /* LF */
            character = Serial$Input; /* ACK? */
            if character <> ACK then do;
                call console$output(character);
                return;
            end;

/*    Get a buffer ready to send */
            status = Read$sector(file$id, .buffer);
            do while status = 0;
                call Serial$Output(STX);
                checksum = 0;
                do i = 0 to 127;
                    character = buffer(i);
                    checksum = checksum + character;
                    call Serial$Output(character);
                    end;
                call Serial$Output(checksum);
                call Serial$Output(ETX);

/*    Buffer sent OK. Was it received OK */
                character = Serial$Input;
                if character = ACK then do;
                    call console$output(loop$count + '0');
                    loop$count = (loop$count + 1) MOD 10;
                    status = Read$sector(file$id, .buffer);
                    end;
                else do;
                    call Console$Output('?');
                    status = 0; /* Retransmit */
                    end;
                end;
/*    File sent */
            status = Close$file(file$id);
            if status <> 0 then call Print$String(.(cr,lf,
                'Could not close Local file', cr,lf,'$'));
            cal Serial$Output(ETX);
            return;
            end;
        end;
    end Send;
end Send;
```

230879-001

```
:F1:FILEIO.PLM

$debug
File$io: do;

/*    This version contains all of the fileio definitions for ISIS */

$nolist include (:f3:common.lit)
$nolist include (:f3:isis.ext)
declare file$id byte external;

/*    Operating system dependant Console Routines */
Console$Input: Procedure byte public;
    return ci;
end Console$Input;

Console$Output: Procedure (char) public;
    Declare char byte;
    call co(char);
end Console$Output;

Console$Status: Procedure byte public;
    return csts;
end Console$Status;

Print$String: Procedure (string$ptr) public;
    Declare string$ptr pointer,
            text based string$ptr byte;
    do while text <> '$';
        call co(text);
        string$ptr = string$ptr + 1;
    end;
end Print$String;

/*    Operating system dependant file routines */
Open$file: Procedure (file$ptr,mode) byte public;
/*    Return OFFH if file does not exist, otherwise return file ID */
    Declare file$ptr pointer,
            mode byte,
            (aftn, status) word;
    call rename(file$ptr, file$ptr, .status);
    if status = 13 then return OFFH; /* File does not exist */
    call open(.aftn, file$ptr, mode, 0, .status);
    if status = 12 then return file$id;   /* 12 returned if file already open
                                             want it open, so it's ok.
                                          */
    if status <> 0 then return OFFH;
    return low(aftn);
end Open$File;

Create$file: Procedure (file$ptr) byte public;
/*    Return OFFH if file already exists, otherwise return file ID */
    Declare file$ptr pointer,
            (aftn, status) word;
    call rename(file$ptr, file$ptr, .status);
    if status <> 13 then return OFFH; /* File already exists */
```

230879-001

```
:F1:FILEIO.PLM

    call open(.aftn, file$ptr, read$write, 0, .status);
    if status <> 0 then return 0FFH;
    return low(aftn);
    end Create$File;

Read$sector: Procedure (file$id, buffer$ptr) byte public;
    Declare file$id byte,
            buffer$ptr pointer,
            (buffer based buffer$ptr) (1) byte,
            (actual, status, i) word;
    call read(double(file$id), buffer$ptr, 128, .actual, .status);
    if status <> 0 then return 0FFH;
    if actual = 0 then return 0FFH;
    if actual <> 128 then do i = actual to 128;
        buffer(i-1) = ' ';
    end;
    return 0;
end Read$sector;

Write$sector: Procedure (file$id, buffer$ptr) byte public;
    Declare file$id byte,
            buffer$ptr pointer,
            status word;
    call write(double(file$id), buffer$ptr, 128, .status);
    return not (status = 0);
end Write$sector;

Close$file: Procedure (file$id) byte public;
    Declare file$id byte,
            status word;
    call close(double(file$id), .status);
    return not (status = 0);
end Close$File;

end fileio;
```

230879-001

```
:F1:SERIAL.PLM

$DEBUG
Serial$IO$for$iPDS: do;

/* This module contains all of the iPDS specific serial IO routines */

Serial$Status: Procedure byte public;
    return ((input(091H) and 2) = 2);
end Serial$Status;

Serial$Input: Procedure byte public;
    do while not Serial$status;
        /* Wait */
    end;
    return (input(090H));
end Serial$Input;

Serial$Output: Procedure (char) public;
    Declare char byte;
    do while ((input(091H) and 1) = 0);
        /* Wait */
    end;
    output(090H) = char;
end Serial$Output;

Serial$Control: Procedure (value) public;
    Declare value byte;
    output(091H) = value;
end Serial$Control;

end Serial$IO$for$iPDS;
```

SOURCE CODE FOR RECV AND SEND NETWORK FILES

```
:F1:RECV.PLM

recv: do;

/*    This is an ISIS utility ptogram for use with a Remote Computer. */

/*    This utility will run on an ISIS cluster board which is connected
      to a remote computer rather than a dumb terminal.
*/

$nolist include(:f3:common.lit)
$nolist include(:f3:isis.ext)

Declare buffer(128)    byte;
Declare (actual, status, aftn) word;
Declare (i, j, checksum, character) byte;

/*    Read the remainder of the command line */
call read(1, .buffer, 128, .actual, .status);
```

230879-001

```
:F1:RECV.PLM

/*    Is the requested ISIS file available */
j = 0;
do i = 0 to 2;      /* Skip "RECV <FILENAME> FROM" in the command word.
                       Need to get the file on the NDS-II system.
                       Don't need to check for syntax - it is already done
                       by the program on the remote computer.
                    */
    do while buffer(j) <> ' ';
        j = j + 1;
    end;
    do while buffer(j) = ' ';
        j = j + 1;
    end;
end;
call open(.aftn, .buffer(j), 1, 0, .status);
if status <> 0 then do;
    call write(0,.(cr,lf,' NDS-II file does not exist',cr,lf), 32, .status);
    call exit;
end;

/*    File is OK */
/*    Get the first buffer of information */
call read(aftn, .buffer, 128, .actual, .status);

do while actual <> 0;
/*    and send it */
    if actual <> 128 then do i = actual to 128;
        buffer(i-1) = ' ';
    end;
    call co(STX);
    checksum = 0;
    do i = 0 to 127;
        call co(buffer(i));
        checksum = checksum + buffer(i);
    end;
    call co(checksum);
    call co(ETX);

/*    Did the Remote Computer receive this OK */
    character = ci and 7FH;

    if character = EOT then call exit;        /* Remote error */
    if character = ACK then call read(aftn, .buffer, 128, .actual, .status);

/*    otherwise assume a transmission error and resend */
end;

/*    Arrive here when the complete file has been sent */
call close(aftn, .status);
call co(ETX);
call exit;

end recv;
```

230879-001

```
:F1:SEND.PLM

send: do;

/*    This is an ISIS utility ptogram for use with a Remote Computer. */

      This utility will run on an ISIS cluster board which is connected
      to a remote computer rather than a dumb terminal.
*/

$nolist include(:f3:common.lit)
$nolist include(:f3:isis.ext)

Declare buffer(128)    byte;
Declare (actual, status, aftn) word;
Declare (i, j, match, count, checksum, received$checksum, character) byte;
Declare delimit(4) byte data (' TO ');

Uppercase: Procedure (char) byte;
Declare char byte;
    if ((char >= 'a') AND (char <= 'z')) then return (char - 20h);
    return char;
end uppercase,

/*    Read the remainder of the command line */
call read (1, .buffer, 128, .actual, .status);

do i = 0 to actual-1;
    if buffer(i) = ' ' then do;
        match = true;
        do j = 0 to 3;
            if uppercase(buffer(i+j)) <> delimit(j) then match = false;
        end;
        if match then do;

/*    We have found the filename */
            i = i + 3;
            do while buffer(i) = ' ';
                i = i + 1;
            end;
            call open(.aftn, .buffer(i), 3, 0, .status)

/*    Did the file already exist? */
                call read(aftn, .buffer, 1, .actual, .status);
                if actual = 1 then do;
                    call write(0.,(cr,lf,:'NDS-II file already exists',cr,lf),
                    30, .status);
                    call exit;
                end;

/*    File is OK, tell Remote Computer to proceed */
                call co(ACK);

/*    Receive the first buffer of information */
                do forever;
                    character = ci;    /* STX or ETX */
```

:F1:SEND.PLM

```
                if character = ETX then do;
                    call close(aftn,.status);
                    call write (0,.(cr,lf,'File transfer complete',cr,lf), 26,
                    .status);
                    call exit;
                end;
                checksum = 0;
                do i = 0 to 127;
                    buffer(i) = ci;
                    checksum = checksum + buffer(i);
                end;
                received$checksum = ci;
                character = ci;             /* ETX */
                if received$checksum <> checksum then call co(NAK);
                else do;
                    call write(aftn, .buffer, 128, .status);
                    call co(ACK);
                end;
            end;
        end;    /* No match, keep looking */
    end;
end;        /* End of line */
call write(0, .(cr, lf,CR,LF, 'Missing <TO>. Correct syntax is:',cr,lf,
        'SEND <local_filename> TO <NDS-II_filename>',cr,lf), 84, .status);
call exit;

end send;
```

230879-001

CHANGES TO SERIAL.PLM TO RUN ON A SERIES III

```
:F1:SERIAL.S2

$DEBUG
Serial$IO$for$SII: do;

/* This module contains all of the SII specific serial IO routines */

Serial$Status: Procedure byte public;
    return ((input(0F7H) and 2) = 2);
end Serial$Status;

Serial$Input: Procedure byte public;
    do while not Serial$status;
        /* Wait */
    end;
    return (input(0F6H));
end Serial$Input;

Serial$Output: Procedure (char) public;
    Declare char byte;
    do while ((input(0F7H) and 1) = 0);
        /* Wait */
    end;
    output(0F6H) = char;
end Serial$Output;

Serial$Control: Procedure (value) public;
    Declare value byte;
    output(0F7H) = value;
end Serial$Control;

end Serial$IO$for$SII;
```

230879-001

CHANGES TO FILEIO.PLM AND REMOTE.MKE TO RUN UNDER CPM

```
:F1:FILEIO.CPM

$DEBUG
CPM$Interface$Library: do;

/* This module contains all of the definitions for FILEIO.EXT for CPM80 */

$nolist include(:f3:common.lit)
$list

Declare bdos$jump address data ( 5 ); /* Set up the address of where to
       go in memory to get to the CPM BDOS routines. This is done
       by using a call by address with parameters of which bdos routine
       and parameters to the routine. This is a clumsy way to do it because
       there is no way to read the return value of the routine. So
       as you will see in the procedure bdos, how the compiler is fooled into
       generating the asm code for a return value.
    */

/* PLM80 Declarations for CPM80 functions */
BDOS: Procedure (type, parameter) byte;
    Declare type byte,
            parameter word;
    if 1 = 2 then return 1; /* Let pass 2 of the compiler see a return for the
                               typed procedure. Then pass 3 will see 1 is
                               never = 2, so it will throw out the statement.
                               Now bdos has put the return value in the Acc.,
                               and the calling procedure that called this
                               procedure will get it out of the Acc.
                               Real clumsy but works.
                            */
    call bdos$jump (type,parameter);
end BDOS;

/* Some BDOS calls return a word; to conform to good PLM syntax we use:....*/
BDOSW:Procedure (type, parameter) word;
    Declare type byte,
            parameter word;
    if 1 = 2 then return 1;
    call bdos$jump (type,parameter);
end BDOSW;

/* Some BDOS calls return nothing; to conform to good PLM syntax we use:....*/
BDOSN:Procedure (type, parameter);
    Declare type byte,
            parameter word;
    call bdos$jump (type,parameter);
end BDOSN;

/*    In case of fatal errors */
Exit: Procedure public;
    call BDOSN(0,0);
    end Exit;
```

230879-001

```
:F1:FILEIO.CPM

/*    Operating system dependant Console Routines */

Declare waiting$char byte; /* This is the buffer for the CO,CI and
                              CSTS BDOS routines */
Console$Status: Procedure byte public;
    waiting$char = BDOS(6,0FFH);
    IF waiting$char = 0 THEN return false;
    return true;
end Console$Status;

Console$Input: Procedure byte public;
/* One problem with the ISIS to CPM BDOS conversion is in the console
   inputs, ISIS doesn't echo and CPM does. So, using the CPM
   BDOS direct console I/O to get around the echo problem.
*/
Declare dummy byte;
    do for$ever;
        IF waiting$char <> 0 THEN return waiting$char;
        dummy = console$status;
    end;
end Console$Input;

Console$Output: Procedure (char) public;
    Declare char byte;
    call BDOSN(6,double(char));
end Console$Output;

Print$String: Procedure (string$ptr) public;
    Declare string$ptr pointer;
    call BDOSN(9,string$ptr);
end Print$String;

/*    Operating system dependant file routines */
Declare FCB$free(6) byte initial (1,1,1,1,1,1);
Declare FCB(6) structure (item(36) byte);
Declare Blank$FCB(36 byte data (1,'           ',0,0,0,0,
                                0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
                                0,0,0,0);

Get$FCB: Procedure byte;
    Declare i byte;
    do i = 0 to 5;
        if FCB$free(i) then do;
            FCB$free(i) = false;
            return i;
        end;
    end;
    call Print$String(.(cr,lf,'Too may files open, Program aborted',cr,lf,'$'));
    call exit;
end Get$FCB;

Format$FCB: Procedure(index, file$ptr);
    Declare (index, l) byte,
            file$ptr pointer;
    Declare (text based file$ptr) (l) byte;
```

230879-001

```
:F1:FILEIO.CPM

/*Clear all the required fields */
    call move(36,.blank$FCB,.FCB(index));
/*    Skip over any leading spaces */
    do while text(0) = ' ';
        file$ptr = file$ptr + 1;
    end;
/*    Has a drive been specified ? */
    if text(1) = ':' then do;
        FCB(index).item(0) = text (0) - 'A' + 1;
        file$ptr = file$ptr + 2;
    end;

    file$ptr = file$ptr - 1;
    do i = 1 to 11;
        if text(i) = ' ' then return;
        if text(i) = cr then return;
        if text(i) = '.' then do;
            file$ptr = file$ptr + i - 8;
            i = 8;
        end;
        else FCB(index).item(i) = text(i);
    end;
end Format$FCB;

Set$DMA$Address: Procedure (value);
    Declare value word;
    call BDOSN(26, value);
end Set$DMA$Address;

Select$disk: Procedure (disk);
    Declare disk byte;
    call BDOSN(14,double(disk));
end Select$disk;

Open$file: Procedure (file$ptr,mode) byte public;
    Declare mode byte,
            (file$ptr, FCB$ptr) pointer,
            (index, status) byte;
    index = get$FCB
    call format$FCB(index, file$ptr);
    call Select$disk(FCB(index).item(0)-1);
    FCB(index).item(0) = 0;
    status = BDOS(15,.FCB(index));
    if status = 0FFH then return 0FFH;
    return index;
end Open$File;

Create$file: Procedure (file$ptr) byte public;
    Declare (file$ptr, FCB$ptr) pointer,
            (index, status) byte;
    index = get$FCB;
    call format$FCB(index, file$ptr);
    call Select$disk(FCB(index).item(0)-1);
    FCB(index).item(0) = 0;
    status = BDOS(22,.FCB(index));
```

230879-001

```
:F1:FILEIO.CPM

    if status = 0FFH then return 0FFH;
    return index;
end Create$File;

Read$sector: Procedure (file$id, buffer$ptr) byte public;
    Declare file$id byte,
            buffer$ptr pointer;
    call Set$DMA$Address(buffer$ptr);
    return BDOS(20,.FCB(file$id));
end Read$sector;

Write$sector: Procedure (file$id, buffer$ptr) byte public;
    Declare file$id byte,
            buffer$ptr pointer;
    call Set$DMA$Address(buffer$ptr);
    return BDOS(21,.FCB(file$id));
end Write$sector;

Close$file: Procedure (file$id) byte public;
    Declare file$id byte;
    FCB$free(file$id) = true;
    return BDOS(16,.FCB(file$id));
end Close$File;

end CPM$Interface$library;
```

```
:F1:REMOTE.CPM

; Make file for the ISIS REMOTE program that runs on the iPDS system.
; Author : B. Valentine DSSO Applications Engineering 6/25/83

; First of all define the macros for the MAKE file.
;  Define the subsitution macros :
;   Subsitution macros are used as constant defines. This way, if
;    a major change is made, such as the source code device changes
;    from :f1: to :f2:, the only update to the make file is to change
;    the macro define.

$    SET work_device       to ':f1:'
$    SET 8_bit_exe         to ':f2:'
$    SET 8_bit_lib         to ':f3:'
$    SET database          to ':f4:remote.db'
$    SET svcs_drive        to 'run :f5:'

$    SET plm               to '%"8_bit_exe"plm80'
; Note how macros may be nested and the macro is used with the %'<name>".
$    SET locate            to '%"8_bit_exe"Locate'
$    SET link              to '%"8_bit_exe"link'
$    SET syslib            to '%"8_bit_lib"system.lib'
$    SET plmlib            to '%"8_bit_lib"plm80.lib'
$    SET comlit            to '%"8_bit_lib"common.lit'
$    SET get               to '%"svcs_drive"svcs get %database'
$    SET put               to '%"svcs_drive"svcs put %database'

; Now define the enumeration macros:
$    SET nds2_files        to (recv,send)
$    SET remote_files      to (remote,fileio,serial,rmrecv,rmsend)
$    SET files             to (%all(%nds2_files),%all(%remote_fiels))

; Tell make that we are going to be looking at the files in the database.
$ FOR i in %files
$    svcs %work_device%"i".plm   = %database (%i)
$    svcs %work_device%"i".obj   = %database (%i,,oj)
$END
$ svcs %"work_device"serial.ext  = %database (serial,,cp)
$ svcs %"work_device"fileio.ext  = %database (fileio,,cp)
$ svcs %"work_device"remote      = %database (exec,,oj)
$ svcs %"work_device"send        = %database (send,,cp)
$ svcs %"work_device"recv        = %database (recv,,cp)

; The include files are always required, so get them with the header.
$ HEADER
; Get all the externals and include files from the database
   %get (serial,,cp)     to %"work_device"serial.ext
   %get (fileio,,cp)     to %"work_device"fileio.ext
$ END

; Now start the dependecies

; Set the dependency tree for three separate executable files.
$ IF all >"work_device" remote, %all(%work_device%nds2_files) THEN
$END
```

```
:F1:REMOTE.CPM

$FOR i IN %remote_files
; Build all the object files in the remote program.
$    IF %work_device%"i".obj > %work_device%"i".plm, %comlit,
$    %"work_device"fileio.ext, %"work_device"serial.ext THEN
        %get (%i) to %work_device%"i".plm
        %plm %work_device%"i".plm
        %put (%i,,oj) from %target
$    END
$END

; Check the remote executable file that runs on the iPDS system.
$ IF %"work_device"remote > %all(%work_device%"remote_files".obj),
$ %plmlib, %syslib THEN
$    FOR i in %remote_files
        %get (%i,,oj) to %work_device%"i".obj
$    END
    %link %depend to %"work_device"remote.lnk
    %locate %"work_device"remote.lnk code(103H) symbols lines &
          map print(%work_device"remote.map)
        %put (exec,,oj) from %target
$ END

; Now that the remote program has been checked, check the two programs
; that run on the network.

$FOR i IN %nds2_files
; Check the NDS_II files RECV and SEND.
$    IF %work_device%i > %work_device%"i".plm THEN
        %get (%i) to %depend
        %plm %depend
        %put (%i,,oj) from %work_device%"i".obj
        %link %work_device%"i".obj, %syslib. %plmlib to %work_device%"i".lnk
        %locate %work_device%"i".lnk
        %get (%i,,cp) to :bb: write
        %put (%i,,cp) from %target
$    END
$END
```

230879-001